



Modelowanie i Analiza Systemów informacyjnych (MAS)

dr inż. Mariusz Trzaska
mtrzaska@pjwstk.edu.pl

Wykład 8 Realizacja pozostałych, wybranych konstrukcji UML w obiektowych językach programowania

Zagadnienia

- Omówienie różnych rodzajów ograniczeń występujących w UML
- Implementacja ograniczeń:
 - Dotyczące atrybutów
 - *Unique*
 - *Subset*
 - *Ordered*
 - *Bag*
 - *History*
 - *Xor*
 - Dowolne
- Podsumowanie

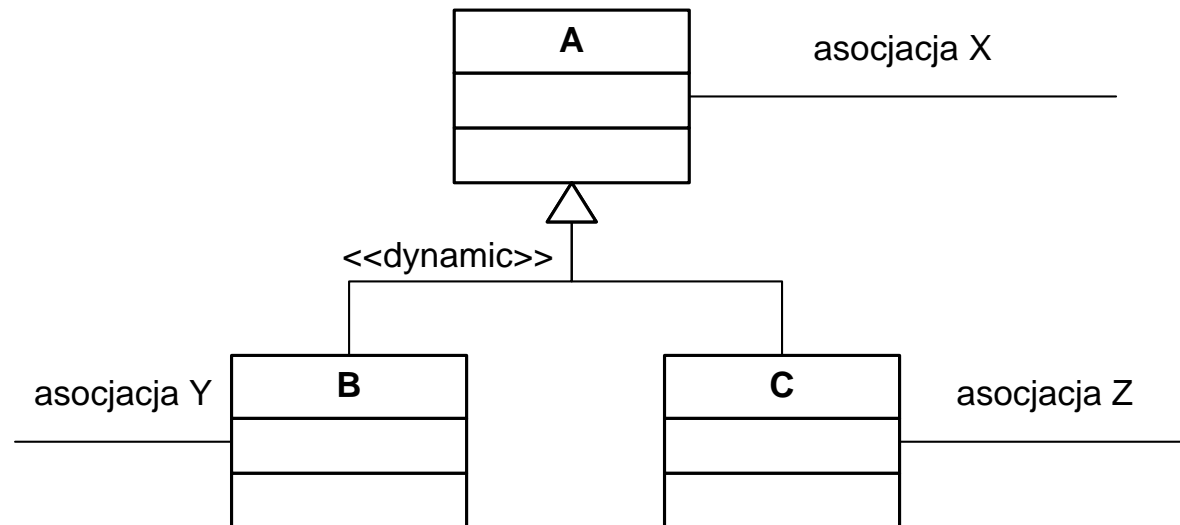
Praca domowa

- Modyfikacja klasy *ObjectPlusPlus* ułatwiająca realizację dziedziczenia dynamicznego.

- Prezentacje studentów...

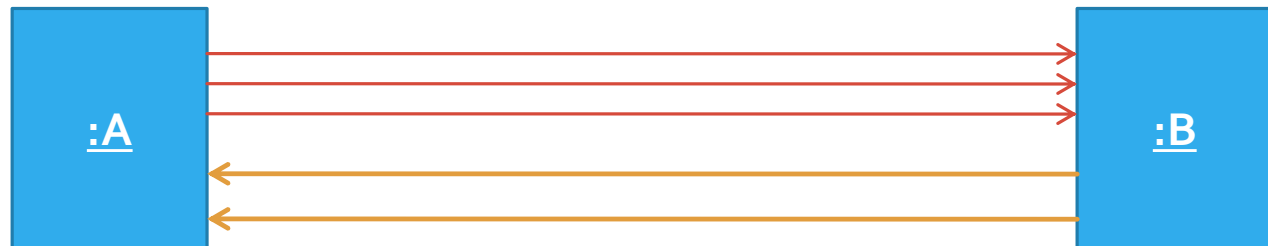
Praca domowa (2)

- Dodatkowe problemy
 - Ze „starego” obiektu powinniśmy skopiować tylko powiązania należące do „nadklasy” (np. A). Nie kopiujemy powiązań należących do jego asocjacji (np. B, C).



Praca domowa (3)

- Dodatkowe problemy – c. d.
 - Aby poprawnie uaktualnić powiązania pokazujące na „nowy” obiekt, potrzebujemy informacji o wzajemnych powiązaniach ról.
 - Która *nazwaRoli* odpowiada *odwrotnaNazwaRoli*,
 - Teoretycznie możemy zastąpić wszystkie wystąpienia (we wszystkich rolach) *obiektDocelowy* pokazujące na „stary” obiekt, ale część z tych uaktualnień jest niepotrzebna, ponieważ część z powiązań „przepada” (patrz poprzedni slajd).



- Powrót do aktualnego wykładu

Ograniczenia w UML

- Jeden z mechanizmów rozszerzalności UML.
- Umożliwiają doprecyzowanie modelu.
- Wykorzystanie:
 - OCL
 - Wyrażenia matematyczne (arytmetyka, logika)
 - Tekst
- Istnieje pewna predefiniowana grupa ograniczeń
- Notacja: {treść ograniczenia}

Rodzaje ograniczeń

- Dynamiczne. Przy sprawdzaniu warunku, istotny jest poprzedni stan elementu, którego ograniczenie dotyczy.
- Statyczne. Przy sprawdzaniu warunku, poprzedni stan elementu, którego ograniczenie dotyczy, nie ma znaczenia.

Rodzaje ograniczeń (2)

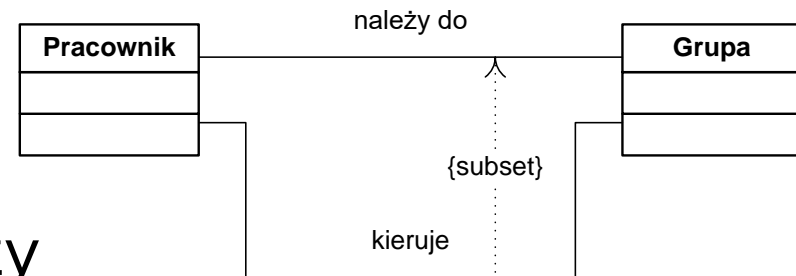
- Przykłady
 - Ograniczenia dynamiczne
 - Pensja nie może zmaleć,
 - Wzrost pensji nie może być większy niż 10%.
 - Ograniczenia statyczne
 - Pensja > 2000
 - Pensja < 5000

Predefiniowane ograniczenia

- {Unique}
 - Nakładane na atrybut w klasie,
 - Zapewnia jego unikalność,
 - Np.:
 - PESEL {unique}
 - Nazwisko {unique}

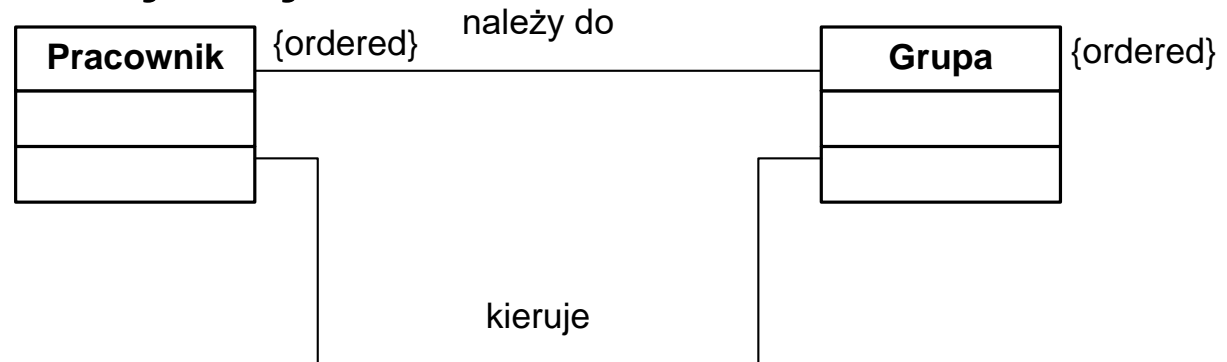
Predefiniowane ograniczenia (2)

- {Subset}
 - Nakładane na dwie asocjacje (agregacje).
 - Aby można było utworzyć powiązanie w ramach asocjacji B („kieruje”), musi już istnieć powiązanie w ramach asocjacji A („należy do”).
 - Obydwie asocjacje powinny być pomiędzy tymi samymi klasami.
 - Obydwa powiązania powinny być pomiędzy tymi samymi obiektami.



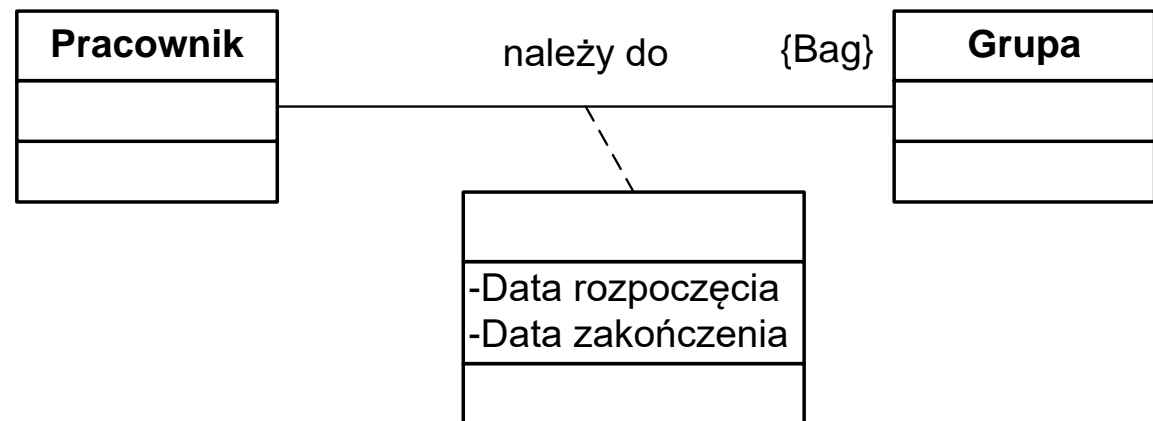
Predefiniowane ograniczenia (3)

- {Ordered}
- Może dotyczyć:
 - Asocjacji. Oznacza, że powiązania są przechowywane (oraz otrzymywane i przetwarzane) w pewnej ustalonej kolejności.
 - Klasy. Obiekty w ekstensji są przechowywane (oraz otrzymywane i przetwarzane) w pewnej ustalonej kolejności.



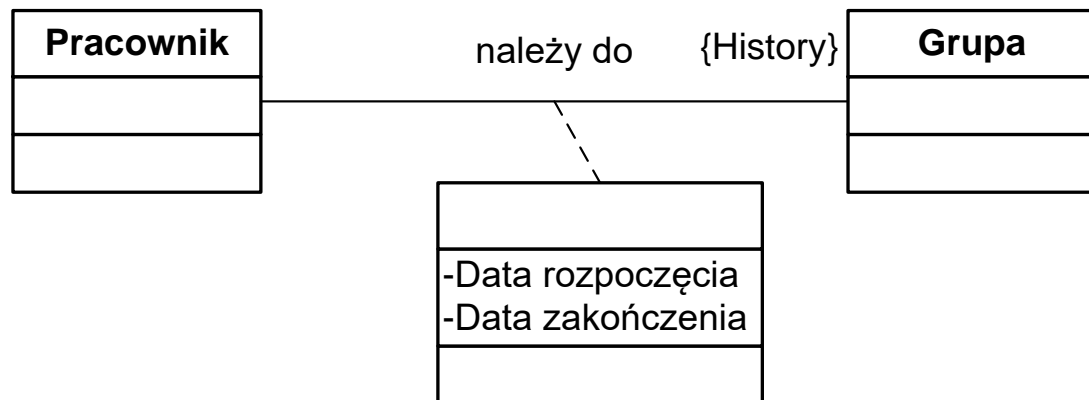
Predefiniowane ograniczenia (4)

- {Bag}
- Umożliwia przechowywanie duplikatów elementów.
- W przypadku asocjacji, oznacza, że może istnieć wiele powiązań pomiędzy tymi samymi obiektami.



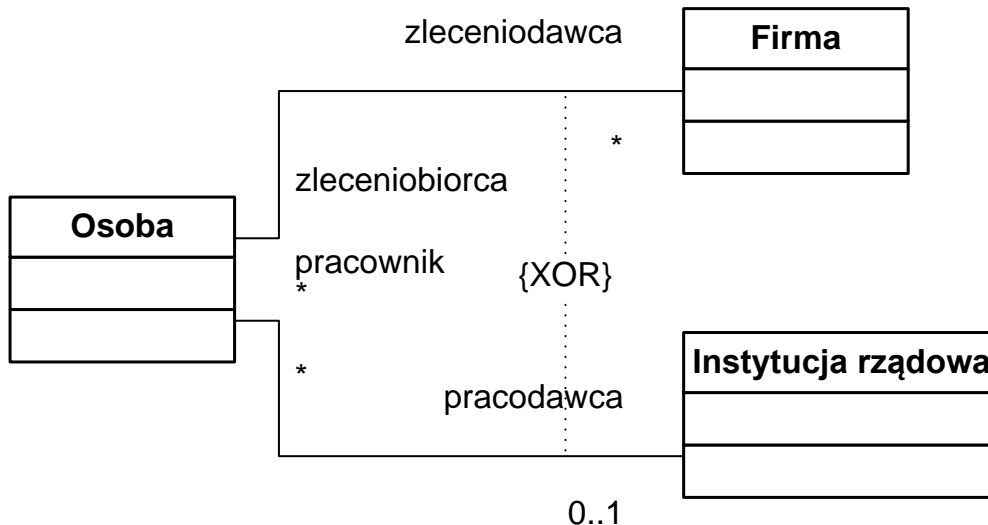
Predefiniowane ograniczenia (5)

- {History}
 - Podobny do ograniczenia {bag}.
 - Umożliwia utworzenie wielu powiązań pomiędzy tymi samymi obiektami.
 - Podkreśla aspekt czasowy opisywanej asocjacji.



Predefiniowane ograniczenia (6)

- {XOR}
- Dotyczy co najmniej dwóch asocjacji.
- Zapewnia, że będzie istniało tylko jedno powiązanie w ramach asocjacji, które ogranicza.



- Jeżeli osoba pracuje w instytucji rządowej to nie może współpracować z firmą.

Ograniczenia UML, a języki programowania

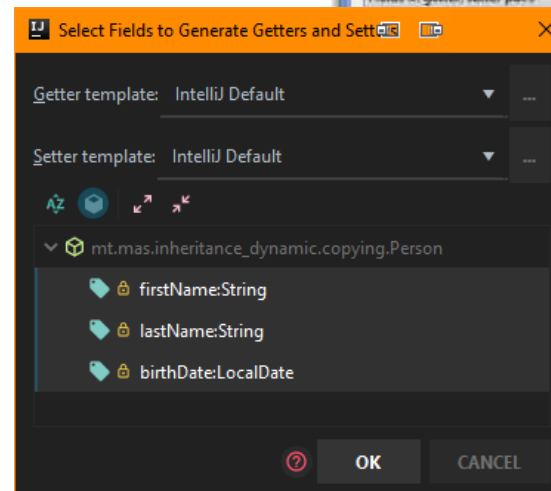
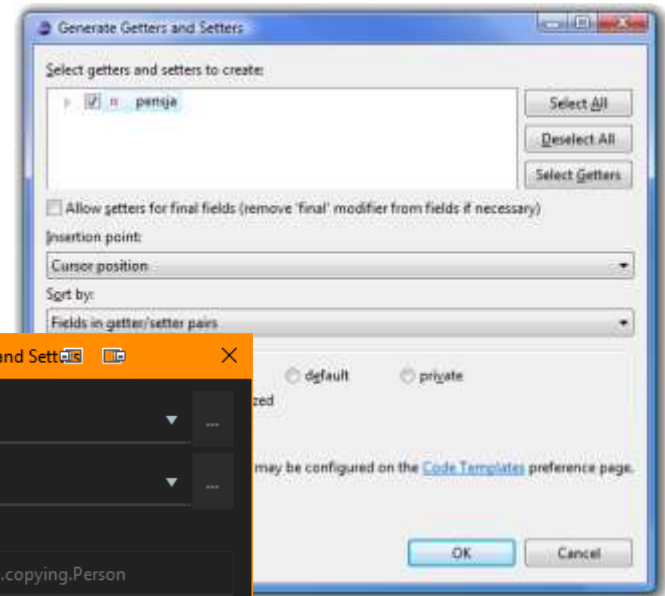
- W popularnych językach programowania:
 - Java
 - C#
 - C++
- ograniczenia nie występują bezpośrednio.
- Możemy je uwzględnić:
 - ręcznie implementując odpowiednie metody.
 - korzystając z dedykowanej biblioteki (np. dla OCL).
- Stopień skomplikowania takich implementacji zależy od rodzaju ograniczeń.

Implementacja ograniczeń dotyczących atrybutów

- Podstawowe założenia
 - Atrybuty w klasie są ukryte (najlepiej jako `private`),
 - Wszelka aktywność związana z atrybutami odbywa się przez dedykowane metody
 - `get...`
 - `set...`
 - Powyższa reguła obowiązuje również wewnątrz klasy w której są atrybuty (choć tego nie da się kontrolować).

Implementacja ograniczeń dotyczących atrybutów (2)

- Współczesne środowiska programistyczne (IDE) wspierają proces tworzenia metod służących do operacji
 - na „ukrytych”
 - atrybutach. M. in.:
 - IntelliJ IDEA
 - Eclipse



Implementacja ograniczeń dotyczących atrybutów (3)

- Uwaga na:
 - nadanie wartości początkowej (może być problematyczne ze względu na procentowe ograniczenia zmian)
 - używanie zdefiniowanych wartości (stałych) i korzystanie z nich we wszystkich miejscach programu (również w czytelnych komunikatach błędów).

Implementacja ograniczeń dotyczących atrybutów (4)

```
public class Employee {
    private float salary;

    public Employee(float salary) {
        this.salary = salary;    // required direct assignment
    }

    public float getSalary() {
        return salary;
    }

    public void setSalary(float salary) throws Exception {
        // Validate all the constraints
        if(salary < this.salary) {
            throw new Exception(String.format("The salary (%s) cannot be decreased!%d", salary));
        }

        if(this.salary * (1d + maxSalaryChangePercentage / 100d) < salary) {
            throw new Exception(String.format("The salary (%s => %s) increase cannot be more then %s%%", this.salary, salary,
                                                maxSalaryChangePercentage));
        }

        if(salary < minimumSalary) {
            throw new Exception(String.format("The new salary (%s) has to be at least %s", salary, minimumSalary));
        }

        this.salary = salary;
    }

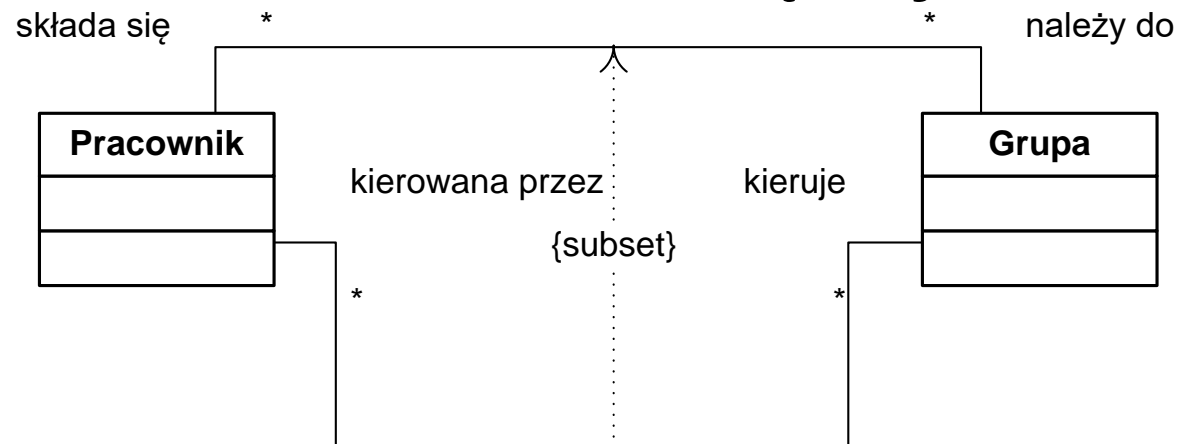
    public final static float minimumSalary = 2000;
    public final static float maximumSalary = 5000;
    public final static int maxSalaryChangePercentage = 10;
}
```

Implementacja ograniczenia {unique}

- Należy zadbać o unikalną (w obrębie ekstensji) wartość konkretnego atrybutu obiektu, np. PESEL.
- Można to osiągnąć modyfikując „setter” oraz np.:
 - Dodając metodę klasową, która przejrzy całą ekstensję i sprawdzi czy podana wartość gdzieś już nie występuje;
 - Dla każdego atrybutu oznaczonego {unique} dodajemy atrybut klasowy (np. Set) przechowujący wszystkie wartości dotychczas występujące.
 - Dodajemy jeden atrybut klasowy typu Map z kluczem będącym nazwą atrybutu oraz wartością przechowującą Set wszystkich wartości ograniczanego atrybutu.
- Zalety i wady powyższych rozwiązań.

Implementacja ograniczenia {subset}

- Powiązania w ramach asocjacji opisanej przez role „składa się”, „należy do” dodajemy „normalnie”.
- Przed dodaniem powiązania w ramach asocjacji opisanej przez role „kierowana przez”, „kieruje”, sprawdzamy czy istnieje powiązanie do dodawanego obiektu w ramach „nadrzędnej” asocjacji.



Implementacja ograniczenia {subset}

(2)

- Zmodyfikujemy klasę *ObjectPlusPlus*
 - Dodamy metodę, która sprawdzi czy istnieje powiązanie do podanego obiektu w ramach danej roli.
- Utworzymy nową klasę *ObjectPlus4*
 - dziedziczącą z *ObjectPlusPlus*
 - enkapsulującą funkcjonalność związaną z realizacją ograniczeń.

Implementacja ograniczenia {subset}

(3)

- Metoda w *ObjectPlusPlus*, która sprawdza czy istnieje powiązanie do podanego obiektu w ramach danej roli.

```
public abstract class ObjectPlusPlus extends ObjectPlus implements Serializable {
    private Map<String, Map<Object, ObjectPlusPlus>> links = new Hashtable<>();

    // [...]

    public boolean isLink(String roleName, ObjectPlusPlus targetObject) {
        Map<Object, ObjectPlusPlus> objectLink;

        if(!links.containsKey(roleName)) {
            // No links for the role
            return false;
        }

        objectLink = links.get(roleName);

        return objectLink.containsValue(targetObject);
    }
}
```


Implementacja ograniczenia {subset}

(4)

- Metoda w *ObjectPlus4*, która dodaje nowe powiązanie uwzględniając ograniczenie {subset}

```
public abstract class ObjectPlus4 extends ObjectPlusPlus {  
  
    public void addLink_subset(String roleName, String reverseRoleName, String superRoleName,  
                               ObjectPlusPlus targetObject) throws Exception {  
        if(isLink(superRoleName, targetObject)) {  
            // There is a (super) link to the added object in the super role  
            // Create the link  
            addLink(roleName, reverseRoleName, targetObject);  
        }  
        else {  
            // No super link ==> exception  
            throw new Exception("No link to the '" + targetObject + "' object in the ," +  
                                superRoleName + "' super role!");  
        }  
    }  
  
    // [...]  
  
}
```

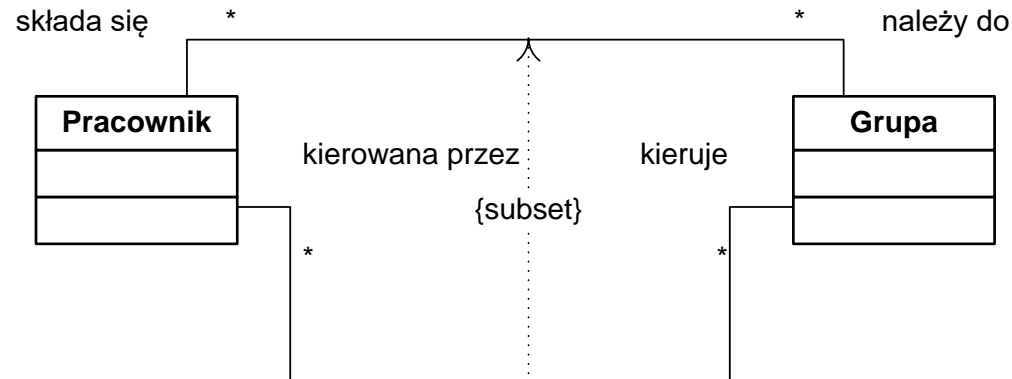
Implementacja ograniczenia {subset}

(5)

- Przykład użycia – wersja automatyczna

```
private static void testSubset_auto() {  
    var employee = new Employee("John", "Smith");  
    var group = new Group("Group no 1");  
  
    try {  
        // Add the ordinary link  
        employee.addLink(Employee.roleBelongsTo, Group.roleConsistsOf, group);  
  
        employee.addLink_subset(Employee.manages, Group.roleManagedBy, Employee.roleBelongsTo, group);  
  
        // Show links info  
        employee.showLinks(Employee.roleBelongsTo, System.out);  
        employee.showLinks(Employee.manages, System.out);  
    } catch (Exception e) {  
        // [...] }}  
}
```

Employee links, role 'belongs to':
Group: Group no 1
Employee links, role 'manages':
Group: Group no 1



Implementacja ograniczenia {subset}

(5)

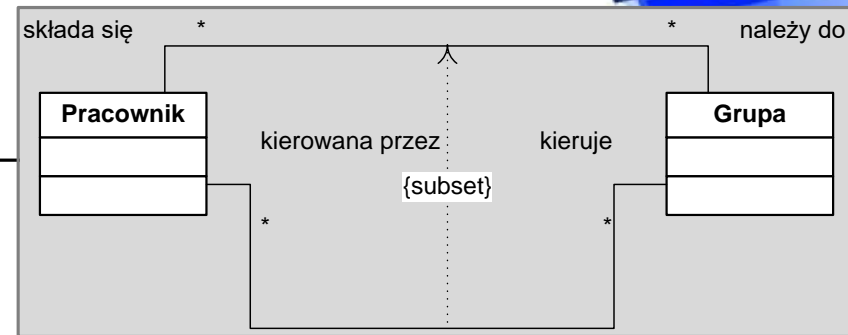
- Przykład użycia - ręczne

```
private static void testSubset_manual() {
    var employee = new Employee("John", "Smith");
    var group = new Group("Group no 1");

    try {
        // Add ordinary link
        employee.addLink(Employee.roleBelongsTo, Group.roleConsistsOf, group);

        // Check if there is a super link
        if(employee.isLink(Employee.roleBelongsTo, group)) {
            // There is a super link => add a subset link
            employee.addLink(Employee.manages, Group.roleManagedBy, group);
        }
        else {
            // No super link
            System.out.println("No super link for the role: " + Employee.roleBelongsTo);
        }

        // Show links info
        employee.showLinks(Employee.roleBelongsTo, System.out);
        employee.showLinks(Employee.manages, System.out);
    } catch (Exception e) {
        // Error
        e.printStackTrace();
    }
}
```



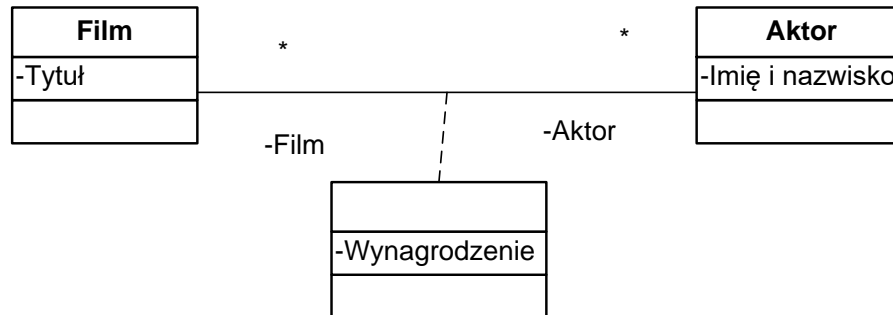
```
Employee links, role 'belongs to':
  Group: Group no 1
Employee links, role 'manages':
  Group: Group no 1
```

Implementacja ograniczenia {ordered}

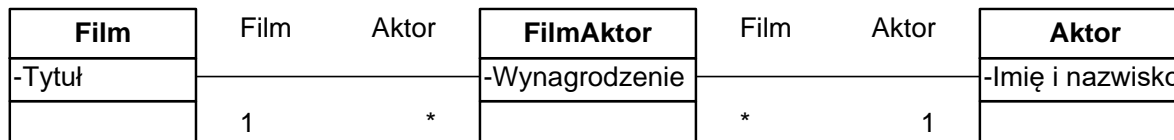
- Dla asocjacji
 - Musimy użyć kontenera gwarantującego kolejność przechowywanych elementów. W przypadku:
 - kolejności dodawania - np. List
 - Indywidualnie zdefiniowanej kolejności, np. TreeSet razem z odpowiednim komparatorem (Comparator).
- Dla ekstensji
 - Analogicznie jak dla asocjacji.
- Klasa *ObjectPlusPlus* domyślnie używa klasy `ArrayList` dla ekstensji (kolejność gwarantowana) oraz `HashMap` dla powiązań.

Implementacja ograniczenia {bag}

- W przypadku asocjacji, musimy użyć kontenera umożliwiającego przechowywanie duplikatów elementów.
- Duplikaty chcemy przechowywać najczęściej gdy korzystamy z atrybutu asocjacji podającego dodatkowe informacje o „duplikacie”.



- A w takiej sytuacji i tak wprowadzamy klasę pośredniczącą, więc problem „duplikatów” znika.



Implementacja ograniczenia {bag} (2)

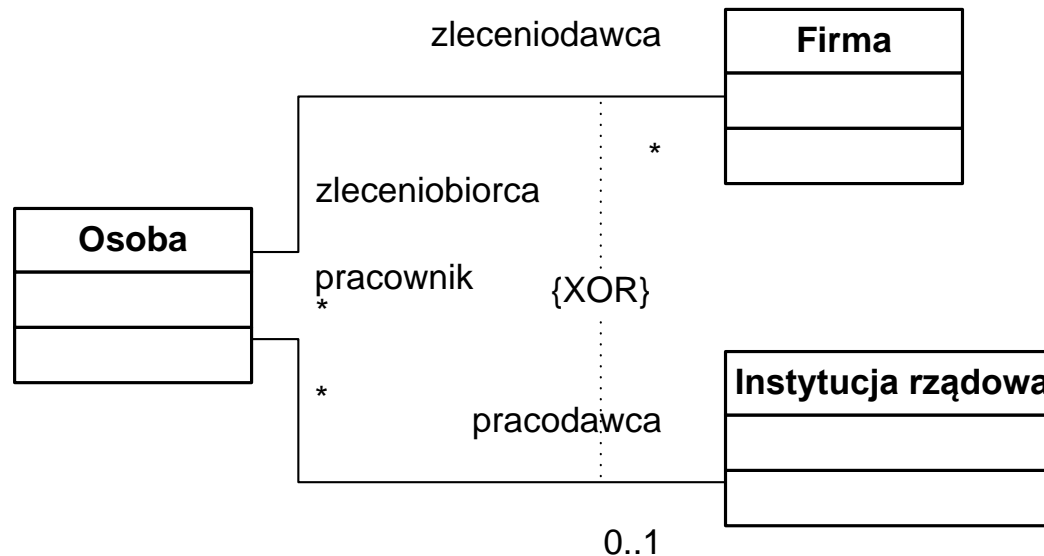
- Gdyby jednak ktoś chciał trzymać duplikaty to trzeba wybrać kontener, który na to pozwala, np. `ArrayList`.
- Klasa *ObjectPlusPlus* domyślnie używa klasy `HashMap` dla powiązań, która pilnuje unikalności kluczy. Można dodawać duplikowane wartości, ale z unikalnymi kluczami.

Implementacja ograniczenia {history}

- Czy użycie ograniczenia {history} oznacza jakieś szczególne konsekwencje (w porównaniu z {bag})?
- Wydaje się, że nie.
- Wniosek: Implementacja ograniczenia {history} jest analogiczna do implementacji ograniczenia {bag}.

Implementacja ograniczenia {XOR}

- Musimy zadbać aby w zdefiniowanej grupie asocjacji (ról) występowało tylko jedno powiązanie.
- Wykorzystamy odpowiednio zmodyfikowaną klasę *ObjectPlus4*.



Implementacja ograniczenia {XOR} (2)

```
public abstract class ObjectPlus4 extends ObjectPlusPlus {  
    private List<String> rolesXOR = new LinkedList<>();  
  
    public void addXorRole(String xorRoleName) {  
        rolesXOR.add(xorRoleName);  
    }  
  
    public void addLinkXor(String roleName, String reverseRoleName, ObjectPlusPlus targetObject)  
throws Exception {  
        if(rolesXOR.contains(roleName)) {  
            // The currently adding role is XOR'ed  
  
            // Check if there is a link for XOR'ed roles  
            if(isXorLink()) {  
                throw new Exception("There is a link for a XOR roles!");  
            }  
  
            // There is no link ==> add a link using an already existing method from a super class  
        }  
  
        // Add the link  
        super.addLink(roleName, reverseRoleName, targetObject);  
    }  
    // [...]  
}
```

Implementacja ograniczenia {XOR} (3)

```
public abstract class ObjectPlus4 extends ObjectPlusPlus {  
  
    private List<String> rolesXOR = new LinkedList<>();  
  
    // [...]  
  
    private boolean isXorLink() {  
        for(String role : rolesXOR) {  
            if(this.anyLink(role)) {  
                return true;  
            }  
        }  
  
        return false;  
    }  
}
```

- Ewentualnie do zrobienia:
 - Uwzględnienie ograniczenia {xor} w przypadku dodawania powiązań z „drugiej strony”

Implementacja dowolnych ograniczeń

- W przypadku dowolnych, innych ograniczeń, np. pisanych „zwykłym tekstem”
 - Niestety trzeba zrozumieć „co autor miał na myśli”,
 - Zaimplementować to wykorzystując odpowiednie metody.
 - Czasami, może być wymagana zmiana podejścia do tworzenia programu, np. zastosowanie ortodoksyjnej hermetyzacji.
 - Brak gotowych rozwiązań.

Podsumowanie

- W popularnych językach programowania ograniczenia nie występują bezpośrednio.
- Pewne ich rodzaje można implementować korzystając z gotowych rozwiązań, np.
 - Dotyczące atrybutów,
 - Predefiniowane: {subset}, {xor}.
- W ogólnym przypadku (ograniczenia pisane zwykłym tekstem) konieczna jest ich ręczna implementacja.

Pliki źródłowe

- Pobierz pliki źródłowe do wszystkich wykładów MAS



<http://www.mtrzaska.com/plik/mas/mas-source-files-lectures>